

03-13-00

A

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventorship..... Allen
Applicant..... Microsoft Corporation
Attorney's Docket No. MS1-489US
Title: Session-State Manager

TRANSMITTAL LETTER AND CERTIFICATE OF MAILING

To: Commissioner of Patents and Trademarks,
Washington, D.C. 20231

From: Kasey C. Christie (Tel. 509-324-9256; Fax 509-323-8979)
Lee & Hayes, PLLC
421 W. Riverside Avenue, Suite 500
Spokane, WA 99201

The following enumerated items accompany this transmittal letter and are being submitted for the
matter identified in the above caption.

- 1. Specification-title page, plus 48 pages, including 50 claims and Abstract
2. Transmittal letter including Certificate of Express Mailing
3. 7 Sheets Formal Drawings (Figs. 1-8)
4. Return Post Card

Large Entity Status [x]

Small Entity Status []

Date:

3-9-2000

By:

Kasey C. Christie
Kasey C. Christie
Reg. No. 40,559

CERTIFICATE OF MAILING

I hereby certify that the items listed above as enclosed are being deposited with the U.S. Postal
Service as either first class mail, or Express Mail if the blank for Express Mail No. is completed below, in
an envelope addressed to The Commissioner of Patents and Trademarks, Washington, D.C. 20231, on the
below-indicated date. Any Express Mail No. has also been marked on the listed items.

Express Mail No. (if applicable)

EL47237000

Date:

03/09/2000

By:

Lori A. Vierra
Lori A. Vierra

jc678 U.S. PTO

09/522063

03/09/00

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Session-State Manager

Inventor:
Joshua Allen

ATTORNEY'S DOCKET NO. MS1-489US

TECHNICAL FIELD

This session-state manager relates to tracking session-state information in a computer communications network environment.

BACKGROUND

The communications protocol used in a typical mainframe environment maintains state information about each user's session. A session is an active connection between two computers. For example, a user begins a session when the user logs into and is authorized by a mainframe computer from a workstation.

Communication between the user's workstation and the mainframe is maintained during the session. Every time the user communicates with the mainframe, the mainframe must confirm who the user is and whether the user is currently logged-on. The communications protocol used between the mainframe and the user's workstation facilitates this session-state tracking.

Identification, validation, and authorization are main reasons for tracking a user's session state information. The mainframe identifies the user.

If the identified user is already logged-on, the user is validated. It is not desirable to force a validated user to logon each time there is any communication. Other than being frustrating, this would be extremely inefficient. Once the user has been identified and the user's logon is validated, the mainframe manages her access to available resources.

Typically, users are not authorized to have access to all available resources. Instead, each user is given authorization to access a specific set of resources. Maintaining readily available session-state information makes it easy to identify, validate, and authorize users.

1 Part of the core problem with all of session-state management techniques is
2 identifying the user. One way to identify the user is to employ the unique address
3 of the client computer that the user is using. Another way is to use a unique user
4 identification indicator (UserID). The unique UserID is a popular choice because
5 it is personal, reliable, and portable. Using the “unique” address of a client
6 computer is often not a desirable option because, depending on the exact network
7 configuration, the address might not be personal, it might not always be unique,
8 and might not be portable.

9 The universal session-state management issues include:

- 10 • User Identification—Who is the user, and is the user really who she claims
11 to be?
- 12 • Logon Validation—Is the user currently logged on?
- 13 • Logon Expiration —How long has it been since last contact with the user?

14
15 Session-state information identifies the user and validates that the user is currently
16 logged-on. Session-state information is tracked during a session.

17 A session begins when the user first contacts the server. User
18 authentication is the first task in the session. Typically, a user provides a user
19 name and a password. If the user name and password match accordingly in the
20 authentication database, then a user begins a session and session-state tracking is
21 initiated.

22 In addition to user identification and logon validation, it is often desirable
23 to track the last time this particular user was active. The system may be
24
25

1 programmed to automatically disconnect a user if there has been no activity for a
2 specified time block. This is called "time-out" or "logon expiration."

3 Logon expiration is optional. However, most systems utilize some sort of
4 session time-out tracking. This is particularly true in communications networks
5 that employ communications protocols that do not inherently track session-state
6 information. These networks are called "stateless" Most asynchronous
7 communications networks are stateless.

8 Stateless networks have no mechanism for monitoring the state of a user's
9 session. They can only gauge a user's desire to maintain a session by active
10 communications from the user. A stateless network cannot determine if the user
11 has turned off their computer and gone home for the day. In other words, a user of
12 a stateless network can discontinue use without "logging-off."

13 Time-out is the mechanism utilized to automatically "log-off" a user after a
14 given period of inactivity. Typically, a user-associated record is kept that indicates
15 the last time there was contact from the user. If the "time-out" time block passes
16 between user activities, then the user is timed-out. If she returns, the user is forced
17 to logon again.

18 Reasons for session time-out include security and releasing of resources.
19 The time-out feature adds security by making it less likely for someone to
20 impersonate another by using the other's computer. Session timeout also
21 facilitates the release of resources allocated to the user during their session so that
22 such resources can be used by others.

23 Other user-related information can be tracked as session-state information,
24 such as the contents of an electronic "shopping cart," purchase history,
25

1 personalization factors, and the like. However, these are not as vital for security
2 and identification as are the universal session-state management issues.

3 Limited-access sites on the World Wide Web ("Web") have introduced the
4 need for tracking session states over the Internet. This is particularly true in the
5 realm of Internet commerce, which is also called "e-commerce", "e-business",
6 "on-line business", "Web-business", and the like. It is vitally important that an e-
7 business Web site have a reliable mechanism in place for users to establish and
8 maintain a session.

9 However, the communications protocol that is typically used with the Web
10 is stateless. Standard Web protocols (such as Hyper Text Transfer Protocol
11 "HTTP") can be thought of as a request/response pair. That is, a client or user
12 makes a request that is filled by a server. After the request/response is complete, a
13 Web server, cannot tell if a user intends to make further requests. Thus, in secure
14 sites, a server cannot tell if a user is still logged in; has moved on to other sites, or
15 has turned off the computer and gone home.

16 In stateless protocols, with each communication, the Web server must
17 check on the state of the user's session. Since the protocol does not assist with
18 this, techniques have been developed to accomplish session-state checks. These
19 techniques typically involve saving state information somewhere and examining
20 that information with each exchange.

21 Fig. 1 shows an example of a typical e-commerce network configuration
22 100. In order to track the session state information of the user, a typical Internet
23 configuration includes three session-state storage tiers 102, 104, and 106. At 102,
24 session-state storage tier 1 includes a client 110. The client 110 is a computer
25

1 (such as a laptop, desktop, or special purpose Internet device) that is capable of
2 running a Web browser. An example of a Web browser is "Internet Explorer"
3 from Microsoft Corporation. The client 110 is connected via its Internet Service
4 Provider (ISP) 112 to the Internet 114.

5 At 104, session-state storage tier B includes one or more Web servers 120
6 connected to the Internet 114. At 106, session-state storage tier C includes a Web
7 database 130 connected to the Web servers 120. Session-state information can be
8 stored at any one of, or any combination of, these three tiers. Each option or
9 combination of options has drawbacks and limitations, such as scalability,
10 reliability, resource efficiency, loss of centralized control, and security.

11 **Tier A (Storing on the client):** The Web server 120 may pass session state
12 information to the client 110 for the client to store locally. The information is
13 stored within tier A. Each time the client sends info to the Web server, it passes
14 along the stored session-state information. The server updates the information and
15 passes it back to the client. This exchange can be accomplished using hidden form
16 fields, universal resource locators (URLs), cookies, or other similar techniques.
17 Those who are skilled in the art are familiar with the available techniques to
18 exchange administrative information between a Web server and a Web browser.

19 The disadvantages of storing session-state information at Tier A include:
20 lack of speed, inefficiency, unreliability, and lack of security. This technique
21 generates a large amount of administrative data that is flowing back and forth with
22 each communication. In addition, the Web browser must send all session data on
23 each request to enable the Web server to get the session-state information. This
24 reduces the effective communications speed and efficiency.

1 This technique is unreliable and unsecure because the user's session-state
2 information is stored outside of the control of the Web server. Since it is stored
3 with the client, the Web server must trust the information that it receives.
4 However, the information that the server receives can be accidentally incorrect
5 (because of data corruption on the client) or it can be purposefully incorrect
6 (because of an attempted security violation).

7 To avoid security disasters where users masquerade as other users, many of
8 these Tier A techniques encrypt the session-state information. To be more
9 specific, they use two-way encryption schemes, which are schemes where
10 encrypted information can be easily recovered by decryption. A server encrypts
11 session-state information and passes it to the browser. The browser stores it and
12 returns it to the server upon subsequent communications, where it is decrypted,
13 used, and updated by the server before being re-encrypted and returned to the
14 client. This again adds to the overhead and inefficiency.

15 Although encryption makes a security attack less likely, it does not
16 eliminate it. Since these techniques require the use of two-way encryption, the
17 browser-stored, session-state information can be decrypted. The door may be
18 locked, but it is not sealed.

19 **Tier B (Storing on the Web server):** To eliminate the need to encrypt and
20 decrypt the session-state information with each request, the Web server 120 may
21 simply store all session-state information at the Web server itself. The information
22 is stored within Tier B 104. This is the easiest session-state information storage
23 solution.

24 The major drawback to this technique is scalability. E-commerce Web sites
25 are generally hosted by a family of Web servers that work together to balance the

1 load. This balance is accomplished by gatekeepers called “load balancers.” This
2 family of Web servers with a load balancer is often referred to as a “Web farm.”

3 Load balancers typically work by passing an incoming request to any server
4 that is available to process it. Thus, different Web servers in the farm may service
5 subsequent requests from the same user.

6 If session-state information stored at Tier B (on the Web servers), then each
7 server must have access to the session-state information of each currently logged-
8 in user. This is because any server may receive a request from any server at any
9 time. Therefore, the servers must be able to communicate with each other and
10 share session-state information.

11 With session-state information storage in Tier B, each server must maintain
12 a copy of session-state information for all users. Alternatively, groups of servers
13 can service designated groups of users. Therefore, all servers within a given group
14 must maintain a copy of the session-state information for their designed users.

15 The copies of the users’ session-state information are updated whenever
16 that information changes. This communication overhead and storage of redundant
17 information is inefficient. These inefficiencies are not terribly great when dealing
18 with one or two Web servers. However, this overhead can severely impact the
19 performance of a Web site when a Web farm includes more than a few servers.
20 Therefore, this technique fails as the scale increases.

21 **Tier C (Storing on the Web database):** To overcome the inefficiencies of
22 a Tier B solution, a central database may be to store information once. The Web
23 database 130 stores all session-state information for all of the Web servers of the
24 farm to access as needed.
25

1 Using a centralized database ameliorates the scalability problem, but it does
2 not eliminate it. In order to handle increases in workload, the Web database must
3 grow with the Web farm to support the management of additional session-state
4 information of new users.

5 Presently, storing session-state information in Tier C (on the Web database
6 130) is the most common approach to solving the problems of tracking users'
7 session-state information over the Web. Fig. 1 shows an example of this approach.
8 The Web database 130 stores a user's universal session-state information in block
9 140. This block 140 includes "user identification" 142, "current logon status"
10 144, and "time of last contact" 146.

11 Available Tier C techniques of tracking session-state information in Web
12 farms rely on either database-based sessions, queries against a LDAP (Lightweight
13 Directory Access Protocol) service, or an in-memory session state. All of these
14 back-end services running on a Web database will bottleneck as more servers are
15 added to the farm in order to scale. The database or LDAP services on the
16 backend will need to be scaled as the servers scale, and can become very
17 expensive and complex to maintain. Additionally, the request from each Web
18 server to a Tier C service introduces latency and delays to the response to a user's
19 request.

20 Although session-state management techniques exist for use over stateless
21 communications networks (such as the Web), each has drawbacks and limitations.
22 Many address the universal session-state issues of user identification, logon
23 validation, and timeout. A common characteristic of the popular techniques is
24 storing the session-state information at one or more of three tiers (102, 104, and
25

106 of Fig. 1). Storing session-state information at any of the tiers impacts scalability, speed, efficiency, reliability, or security.

SUMMARY

The exemplary implementations of session-state manager described herein overcome the limitations of the existing session-state management techniques for stateless networks. One way that this is accomplished is by *not* storing the session-state information on any tier. Another way is by having a common time tracker to manage session timeouts. Still another way is to use one-way encryption schemes to provide superior security.

Without actually storing session-state information, the described exemplary implementations of session-state manager identify a user, validate the user's current logon state, and determine whether the user's session should expire.

User identification and logon validation are checked by a server in a stateless network by generating a mathematically session-state token and sending that token to a user. Subsequently, the server receives a mathematically session-state token from the user and checks that token. If that token checks out, then the user is allowed continuing access under the same session.

If it doesn't check out, then the user may be forced to start a new session by logging-on again. Alternatively, the server may check to see if the token would check out if it had come at an earlier time block. The session-state tokens are mathematical encoded and are generated using a one-way encryption scheme. Such a one-way encrypted token is scientifically impossible to reverse-engineer.

Furthermore, logon expiration is checked by the server using the same mathematically session-state token. The token is checked to determine whether a

1 predetermined number of time blocks have past. If so, then the server will
2 terminate the user's session.

3 Since the described exemplary implementations of session-state manager
4 uses a mathematical function on each server to determine user identification and
5 logon validity/expiration, it will easily scale as the number of servers in a group of
6 servers ("farm") scale. It also maintains users' session-state information across all
7 servers in a farm without requiring communication between the servers or
8 persisting the session information in memory, a database, or any other server-side
9 store. This allows virtually unlimited scalability without having to build back-end
10 authentication services.

11 **BRIEF DESCRIPTION OF THE DRAWINGS**

12
13 A more complete understanding of exemplary methods and arrangements
14 of the present session-state manager may be had by reference to the following
15 detailed description when taken in conjunction with the accompanying drawings
16 wherein:

17 Fig. 1 is schematic drawing showing the three session-state storage tiers in
18 which session-state information may be stored;

19 Fig. 2 is schematic drawing showing a communications network that may
20 be used in an implementation of the session-state manager;

21 Fig. 3 is schematic drawing showing an exemplary computer that may be
22 used in an implementation of the session-state manager;

23 Fig. 4 is a timeline showing incremental blocks of time that represents
24 examples of time blocks that may be used with an exemplary implementation of
25 the session-state manager;

1 Fig. 5 is a flowchart showing exemplary steps implementing the session-
2 state manager;

3 Fig. 6 is a flowchart showing exemplary steps implementing the session-
4 state manager;

5 Fig. 7 is a flowchart showing exemplary steps implementing the session-
6 state manager;

7 Fig. 8 is a flowchart showing exemplary steps implementing the session-
8 state manager.

9 10 **DETAILED DESCRIPTION**

11 The following description sets forth a specific embodiment of a session-
12 state manager that incorporates elements recited in the appended claims. This
13 embodiment is described with specificity in order to meet statutory enablement
14 and best-mode requirements. However, the description itself is not intended to
15 limit the scope of this patent. Rather, the inventor has contemplated that the
16 claimed session-state manager might also be embodied in other ways, in
17 conjunction with other present or future technologies.

18 The exemplary session-state manager implementation is does not store a
19 user's actual session-state information on any tier in a stateless network. Rather a
20 Web server creates and delivers a one-way encrypted token to a user on a client of
21 that server. Rather than including session-state information, the token incorporates
22 a representation or a digest of the user's session-state information.

23 The server generates a token and sends it to the user. Subsequently, the
24 user sends the token that it received to the server. The server checks out the
25

received token to see if the user has a valid session. The server may update the tokens from time to time.

The exemplary session-state manager is implemented on a server on the World Wide Web ("Web"). The Web server hosts an access-restricted Web site. Since the Web is stateless, the Web protocol does not help track session-state information.

Part of the core problem with all of session-state management techniques is identifying the user. One way to identify the user is to use the unique address of the client computer that the user is using. Another way is to use a unique user identification indicator (UserID).

Some computers now access the Internet via devices called proxies. These proxies provide a single Internet Protocol ("IP") address for multiple users.

Fig. 2 shows a representation of a communications network 200 such as the Internet. The network includes clientA 202, clientB 204, clientC 206, and clientD 208. These clients are any computer that can connect to the Internet. For example, a client may be a special purpose Internet device, or any type of computer such as a desktop PC, a laptop, a palmtop, a Macintosh, or some other computer running a Web browser.

ClientD 208 connects to the Internet 210 through an Internet Service Provider (ISP) 212. When clientD connects through ISP 212, the ISP assigns clientD its own unique IP address. ClientD may be identified by that IP address. ISP 212 represents one or more ISPs.

ClientsA-C connect to the Internet 210 via ISP 220. ClientA 202 and ClientB 204 use "Internet Explorer" by the Microsoft Corporation as its Web browser. ClientC 206 uses some other Web browser.

1 ISP 220 may provide multiple paths to the Internet 210. Many large ISP no
2 longer assign IP addresses to clients. Rather, they use a set of Internet proxies,
3 such as proxy-1 222 and proxy-2 224. Proxies have their own unique IP address.

4 As their name implies, they act as an IP address proxy for the clients. Each
5 time a communications comes from a client, it may use a different proxy. Thus, it
6 will use a different IP address each time.

7 Communications over the Internet 210 to/from ClientsA-C may use a
8 different proxy each time. For example, client A may use proxy-1 222 one time.
9 The next time, it might use proxy-2 224.

10 A popular alternative is to have the user identify herself by an identifier of
11 some kind (UserID). This may be called “user name”, “user-id”, “shopper name”,
12 “shopper-id”, and the like.

13 Fig. 2 shows a typical site that a user might logon using a UserID. A Web
14 farm 240 includes a multitude of separate Web servers, such as Web servers 242,
15 244, 246, and 248. One or all of these Web servers implement the exemplary
16 session-state manager.

17 Scalability is a major advantage of a Web farm. As a site becomes more
18 popular, additional Web servers can be added to the farm to support the additional
19 load. A Web farm typically includes a Web database 250. These databases include
20 central information that is shared by all of the Web servers.

21 In the exemplary implementation of the session-state manager, session-state
22 tokens are generated by the Web server. The Web server can be a single Web
23 server as shown at 130 in Fig. 1 or a Web farm as shown at 240 of Fig. 2. These
24 tokens are passed between the clients and the Web server. Examples of such client
25

are shown at 110 in Fig. 1 and at 202-208 of Fig. 2. Preferably, these tokens are mathematically encoded and are generated by a one-way encryption scheme.

The tokens are encoded by including a value that is mathematically determined. Alternatively, the tokens are encoded by including a value representing another value. For example, the token may be encoded by concatenating several binary values or the token may be encoded by including a value representing an entry into a look-up table. The tokens may be encrypted by including a hash value that is generated by a one-way encryption hash.

The exemplary embodiment of the session-state manager is be integrated into the operation of a Web server. For example, the exemplary embodiment uses one or more COM (Component Object Model) components called from within dynamic pages such as ASP (Active Server Page).

Alternatively, an embodiment of the session-state manager may be implemented as a stream filter, such as ISAPI (Internet Server Application Programming Interface), of the Web server. This stream filter can automatically implement the session-state information functions with no programming or modification to existing Web pages required.

Other types of Web servers may have different mechanisms for providing specific functionality extension through plug-ins, snap-ins, or the like. The embodiments of the session-state manager may be implemented using these extension mechanisms, or may be directly incorporated into a component of the Web server.

Fig. 3 shows a computer that is an example of a computer that may be used with an implementation of the session-state manager. This computer may be a

1 client, a Web server, a Web database, or any computer with a communications
2 network (such as the Internet).

3 As shown in Fig. 3, computer 330 includes one or more processors or
4 processing units 332, a system memory 334, and a bus 336 that couples various
5 system components including the system memory 334 to processors 332. Bus 336
6 represents one or more of any of several types of bus structures, including a
7 memory bus or memory controller, a peripheral bus, an accelerated graphics port,
8 and a processor or local bus using any of a variety of bus architectures.

9 The system memory includes read only memory (ROM) 42 and random
10 access memory (RAM) 340. A basic input/output system (BIOS) 342, containing
11 the basic routines that help to transfer information between elements within
12 computer 330, such as during start-up, is stored in ROM 338.

13 Computer 330 further includes a hard disk drive 344 for reading from and
14 writing to a hard disk, not shown, a magnetic disk drive 346 for reading from and
15 writing to a removable magnetic disk 348, and an optical disk drive 350 for
16 reading from or writing to a removable optical disk 352 such as a CD ROM, DVD
17 ROM or other optical media. The hard disk drive 344, magnetic disk drive 346
18 and optical disk drive 350 are each connected to bus 336 by one or more interfaces
19 354.

20 The drives and their associated computer-readable media provide
21 nonvolatile storage of computer readable instructions, data structures, program
22 modules and other data for computer 330. Although the exemplary environment
23 described herein employs a hard disk, a removable magnetic disk 348 and a
24 removable optical disk 352, it should be appreciated by those skilled in the art that
25 other types of computer readable media which can store data that is accessible by a

1 computer, such as magnetic cassettes, flash memory cards, digital video disks,
2 random access memories (RAMs), read only memories (ROM), and the like, may
3 also be used in the exemplary operating environment.

4 A number of program modules may be stored on the hard disk, magnetic
5 disk 348, optical disk 352, ROM 338, or RAM 340, including an operating system
6 358, one or more application programs 360 (such as a Web browser), other
7 program modules 362, and program data 364. A user may enter commands and
8 information into computer 330 through input devices such as keyboard 366 and
9 pointing device 368. Other input devices (not shown) may include a microphone,
10 joystick, game pad, satellite dish, scanner, or the like. These and other input
11 devices are connected to the processing unit 332 through an interface 370 that is
12 coupled to bus 336.

13 A monitor 372 or other type of display device is also connected to bus 336
14 via an interface, such as a video adapter 374. In addition to the monitor, personal
15 computers typically include other peripheral output devices (not shown) such as
16 speakers and printers.

17 Computer 330 can operate in a networked environment using logical
18 connections to one or more remote computers, such as a Web server 382. Web
19 server 382 typically includes many or all of the elements described above relative
20 to computer 330. In addition, a Web database 384 may be connected to the Web
21 server 382.

22 A logical connection that is not depicted in Fig. 3 is a local area network
23 (LAN) via network interface 386 and a general wide area network (WAN) via a
24 modem 378. Such networking environments are commonplace in offices,
25 enterprise-wide computer networks, intranets, and the Internet.

1 Depicted in Fig. 3, is a specific implementation of a WAN via the Internet.
2 Over the Internet, computer 330 typically includes a modem 378 or other means
3 for establishing communications over the Internet 380. Modem 378, which may
4 be internal or external, is connected to bus 336 via interface 356.

5 In a networked environment, program modules depicted relative to the
6 personal computer 330, or portions thereof, may be stored in the remote memory
7 storage device. It will be appreciated that the network connections shown and
8 described are exemplary and other means of establishing a communications link
9 between the computers may be used.

10 11 Tokens

12 The exemplary implementation of the session-state manager uses session-
13 state tokens, rather than storing session-state information. These tokens are
14 generated by a Web server and sent to a user. These tokens are subsequently
15 received from the user and examined by the server.

16 These tokens are used to identify a user, validate the user's session, and
17 expire the user's sessions. These tokens are created by using information that is
18 traditionally tracked to monitor users' session state. Unlike conventional session-
19 state management where session-state information is stored, the tokens provide
20 advantages in scalability, speed, efficiency, reliability, or security.

21 In the exemplary implementation of the session-state manager, the manager
22 generates tokens based upon specific information and compares the just-generated
23 tokens with tokens received from a user. The exemplary implementation of the
24 token generation function, referred to herein as the GetToken() pseudocode,
25 creates a token using specific information. The exemplary implementation of the

1 token comparison function, referred to herein as the CheckToken() pseudocode,
2 takes a received token and compares it to a freshly generated token to see if they
3 match.

4 5 Time Buckets

6 The exemplary implementation of the session-state manager uses a
7 common timeout-tracking scheme to expire sessions after sufficient time has
8 passed. In this exemplary implementation, time is divided into discrete intervals
9 called "time buckets" or simply "buckets." These time buckets form a quantized
10 time measurement that can be tracked and identified simply by assigning a running
11 count. A Web server (such as Web server 130 in Fig. 1 or the Web servers of Web
12 farm 240 of Fig. 2) keeps a running count of these time buckets.

13 Fig. 4 shows an incremental series of buckets at 400. In this exemplary
14 series of buckets, each bucket is one hour long. Of course, the exact length of
15 each bucket is an implementation detail that can be varied based upon the needs of
16 each implementation. Assuming a fixed number of buckets before timeout occurs,
17 the shorter buckets will lead to a shorter timeout period and the longer buckets will
18 lead to longer timeout period.

19 It is assumed for the example shown in Fig. 4 and for the exemplary
20 implementation described in this document that all of the buckets are of equal
21 length. It is possible for the buckets to have variable lengths. However, if that is
22 the case, then it is preferred for there to be a way so that a bucket's length can be
23 quickly determined.

24 In Fig. 4, the first bucket shown is bucket 402, which begins at 9am and
25 ends at 10am. Within a Web server, bucket 402 is identified by a simple ordinal

1 value such as 2,525. Fig. 4 shows other buckets 404-414 with incremental ordinal
2 values 2,526 through 2,531. These ordinal values may be used to identify a time
3 bucket. They may be referred to as "TimeIDs."

4 Incremental buckets before bucket 402 are not shown. Likewise,
5 incremental buckets after bucket 414 are not shown.

6 It is assumed for the example shown in Fig. 4 and for the exemplary
7 implementation described in this document that the value assigned to each bucket
8 is a simple ordinal value. It is possible for the bucket values (or identifications) to
9 be determined in other ways. However, if that is the case, then there must be a
10 way to identify a bucket's value.

11 In the exemplary embodiment of the session-state manager uses time
12 buckets as shown in Fig. 4 as part of the input to create tokens. This results in a
13 situation where the token that is generated depends on the time bucket when the
14 token is created. When the time bucket changes, a different token will be created.
15 As explained in create detail below, this can be used to test for logon expiration.

16 Suppose the exemplary session-state manager called GetToken() at T1 in
17 the time bucket 406 in Fig. 4. At T2, suppose the exemplary session-state
18 manager calls CheckToken() to see if a particular token is still valid. Because the
19 bucket is still bucket 406, the same token is returned (assuming that no other
20 changes have occurred in the information used to generated the token). T3 is
21 within a new time bucket 408. Therefore, a new token is returned from
22 CheckToken(). This new token is used in subsequent calls to CheckToken().

23 A session times out after X number of full buckets. X is a specified number
24 of one or more. Of course, the exact value for X is an implementation detail that
25 can be varied based upon the needs of each implementation.

1 Because a time bucket change will trigger a token change, activities during
2 any point during a bucket are treated equally. For example, in Fig. 4 the buckets
3 are one hour long. Activity at T1 takes place at 11:01am. Activity at T2 takes
4 place at 11:59am. Activity at T3 takes place at 12:01pm.

5 T1 and T2 activities are fifty-eight minutes apart but would generate the
6 same token (assuming no other changes) because they are within the same bucket.
7 T2 and T3 activities are two minutes apart, but would generate different tokens
8 (assuming no other changes) because they are in different buckets. Therefore, the
9 specified length of the buckets should be adjusted so that it meets the needs of the
10 users and of the Web site.

11 The effect of quantized time buckets on the length of time that a user can
12 have a valid session without timing out falls within a range that can be
13 mathematically determined. Assuming X is the number of buckets specified for
14 the timeout value, T_b is the length of a time bucket, Y is approximately the
15 smallest measurable moment of time relative to the length of the time bucket.

16 The minimum time that a user can have session before a timeout (MinTO)
17 may be calculated using the following formula:

$$MinTO = (X + Y)T_b$$

21 The maximum time that a user can have a session before a timeout
22 (MaxTO) can be calculated using the following formula:

$$MaxTO = (2X - Y)T_b$$

Token Generation

Fig. 5 shows the steps of the exemplary implementation of the session-state manager that may be broadly called “token generation.” At 510, the Web server gets a current time bucket identification indicator (TimeID) that represents a current time bucket.

At 512, the Web server gets the user’s UserID that identifies the user of the client. This UserID may have been supplied by the user or is may be retrieved from a database. The UserID may or may not be equivalent to the “username” used for logon authentication.

At 514, the Web server gets a code key (i.e., “secret string” or “trapdoor key”). This code key is defined data that will be used with the TimeID and the UserID so that it is more difficult to decode the encoded token and determine what the TimeID and UserID. This code key may be statically or dynamically designated. If the code key is dynamically designated, it is preferable that code key be tracked carefully so that compared tokens are based upon the same code key.

At 516, the Web server combines the UserID, TimeID and the code key to get the encoded session-state token. In the exemplary implementation, the combining is accomplished by concatenating the UserID, TimeID and the code key. At 518, the Web server encrypts the encoded session-state token to produce an encrypted, session-state token. The encryption preferable uses a one-way encryption scheme, such as a MD5 hash. The session-state token is N bits of the result of the cryptographic hash.

Assuming that $T_{encoded}$ is the encoded token; *UserID* is the identifying name of a user; *TimeID* is a time bucket identifier; and *code key* is defined data (such as

1 a text string), the generation of the encoded token of the exemplary embodiment
2 may be represented by this formula:

$$T_{encoded} = UserID + TimeID + code\ key$$

3
4
5
6 Assuming that $T_{encrypted}$ is the encrypted token; $N[]$ is a function the takes a
7 given number of bits; and $H[]$ is a cryptographic hash function, the generation of
8 the encrypted token of the exemplary embodiment may be represented by this
9 formula:

$$T_{encrypted} = N [H [T_{encoded}]]$$

10
11
12
13 In the $T_{encoded}$ formula, the UserID, TimeID and code key are concatenated
14 (or otherwise combined). In the $T_{encrypted}$ formula, the cryptographic hash (like
15 MD5) of $T_{encoded}$ is performed and N bits of the resulting hash are taken at the
16 encrypted token.

17 In the exemplary embodiment of the session-state manager, a function
18 called GetToken() is used to generate a token. The calling program passes it
19 something preferably unique (e.g., UserID) as an input. GetToken returns an
20 encrypted token, which is unique to the input and can be used to determine what
21 time the user logged in.

An exemplary section pseudocode for the GetToken function is below (the pseudocode is similar to C++ code for a COM implementation):

```

STDMETHODIMP CSessMan::GetToken(BSTR root, BSTR *pbstrToken)
{
    HRESULT hr;
    CComBSTR tmpRoot(root);
    tmpRoot.Append("");
    if (tmpRoot.Length() == 0) {
        CComVariant vtOut("Timeout");
        hr = VariantChangeType(&vtOut, &vtOut, 0, VT_BSTR);
        if (FAILED(hr)) return hr;
        *pbstrToken = ::SysAllocString(V_BSTR(&vtOut));
        return S_OK;
    }
    char tmpOut[33];
    tmpRoot.Append(m_szSecretSeed);
    tmpRoot.Append(GetTimeBucket());
    CalcHash((char *)tmpRoot.m_str, tmpOut, ((int)(tmpRoot.Length())
+ 1) * 2);
    tmpOut[m_HashLength] = 0;
    CComVariant vtOut(tmpOut);
    hr = VariantChangeType(&vtOut, &vtOut, 0, VT_BSTR);
    if (FAILED(hr)) return hr;
    *pbstrToken = ::SysAllocString(V_BSTR(&vtOut));
    return S_OK;
}

```

Encryption Schemes

Two-way encryption schemes are those where the encrypted data can be decrypted. Although there are many different two-way encryption schemes with varying levels of security, all of them allow the encrypted data to be decrypted. Two-way encryption schemes provide a locked door behind which data may be stored. However, that door has a key. If that key falls into the wrong hands, it can be unlocked and the data stored behind the door stolen.

1 One-way encryption schemes are those where the encrypted data cannot be
2 decrypted. *Applied Cryptography* by Bruce Schneier (John Wiley & Sons, Inc.,
3 1994) (p. 27) describes a one-way encryption scheme as one that is “relatively
4 easy to compute but significantly harder to undo or reverse.” It also says that that
5 “hard” means “it would take millions of years to compute....” In general, One-way
6 encryption schemes are far more secure than two-way encryption schemes.

7 Examples of one-encryption schemes that may be used with the exemplary
8 implementation of the session-state manager include a 128-bit MD5 hash, Secure
9 Hash Algorithm (SHA), or any other cryptographically strong one-way hash
10 function. The preferred one-way encryption scheme is fast and produces results
11 that are apparently randomly distributed.

12 Using a one-way encryption hash (like MD5), the session-state token is
13 naturally statistically distributed. This makes it practically impossible for a
14 potential interloper to guess a valid token. The security level can be increased or
15 decreased by changing the strength of the encryption scheme.

16 The security level may be further increased or decreased by selecting the
17 number of bits used from the returned hash. For example, with a 128-bit MD5
18 hash, using only 80 of the returned bits, there is a less than one in one trillion (1 in
19 1,000,000,000,000) probability of the token being correctly guessed. In this
20 exemplary embodiment, only about ten bytes of data are exchanged between the
21 client and server for tracking of session-state information.

22 Since the session-state manager uses a one-way encryption hash that is
23 controlled by the server, it is scientifically impossible for a potential interloper to
24 discover the UserID, TimeID and the code key by “cracking” the encrypted token.
25

1 In addition, it is also practically impossible for a potential interloper to fake client-
2 side information to obtain access to a valid session.

4 Token Initialization

5 Fig. 6 shows the steps of the exemplary implementation of the session-state
6 manager that may be broadly called "token initialization." At 610, a user of a
7 client is authenticated. Typically, a user is asked for her "username" and password
8 at the logon page of the Web site. The user provides the requested information.
9 The authentication server(s) processes the information. The authentication
10 server(s) will allow access or it will deny access.

11 If the user is authenticated and allowed access, an encoded, session-state
12 token is generated by the Web server at 612. This encoded token does not include
13 session state information. Rather, it incorporates representations of the session-
14 state information.

15 In the exemplary implementation, the token is a code that refers (i.e.,
16 points) to an entry into a hash. The token itself does not include the inputs that
17 were used in the one-way encryption hash scheme to generate the resulting code
18 that pointed to the same place in the hash.

19 Alternatively, token may be unencrypted. In other words, the token may be
20 plain text or plain data. However, this plain data may be encoded so that its
21 meaning is not obvious absent additional information. For example, the encoded
22 token may be a plain data reference to a look-up table.

23 A key difference between using an encrypted token and using an
24 unencrypted token is that the latter lends to a discernable pattern due to the look-
25

up table assignments as opposed to the apparently random distribution of the hash scheme. Thus, the look-up table implementation is less secure.

At 614, the token is sent to the client. The client briefly stores this token. Although the client is storing the token, it is not storing session-state information. Again, the token does not include session-state information. Rather, it incorporates representations of session-state information.

When the client makes a request, the client sends that token to the Web server. The data stored on the client is much smaller than with existing techniques that store actual session-state information on Tier A. In this exemplary embodiment, only about ten bytes of are stored on the client.

In the exemplary embodiment of the session-state manager, a user logs onto a Web site. The user must be authenticated.

The pseudocode below shows how a secure Web page on a web site may authenticates a user employing the steps described above and in Fig. 6:

```
` put the following code at the top of the page, before any output
Set logintoken = token from QueryString
Set crypto = Application("CryptoObj")
` previously set to "Orca.Crypto"
set querystring token=crypto.CheckToken(shopperId, logintoken)
if token = "Timeout" then
    Response.Redirect "login.asp"
End if
End if
```

If the user first logs in successfully, a token generation function executes. The logon page calls a token generation function called "GetToken()" and passes the function something preferably unique (e.g., the UserID provided by the user)

1 as an input. GetToken returns an encrypted token, which is unique to the input and
2 can be used to determine what time the user logged in.

3 This token is stored by passing around in the URL. The token could
4 alternately be stored in a client-side cookie, a hidden form field, or the like.

5 6 Token Verification

7 Fig. 7 shows the steps of the exemplary implementation of the session-state
8 manager that may be broadly called “token verification.” At 710, a Web server
9 (which can be a single Web server as shown at 130 in Fig. 1 or a Web farm as
10 shown at 240 of Fig. 2) receives a user identifier (UserID) and a one-way
11 encrypted, session-state token with each transmission from the client.

12 At 712, the Web server generates a one-way encrypted, confirmation
13 session-state token. At 714, the Web server compares the confirmation token with
14 the token that it received in block 710.

15 If the two tokens match, then access to the requested Web page of the
16 server is allowed at 716. The two tokens will match if:

- 17
- the same UserID is used to generated both; and
 - both were generated during the same incremental time block.
- 18

19 That time block would be the one that was current when the confirmation token is
20 generated. As previously discussed, these incremental time blocks are called “time
21 buckets.”

22 If the tokens do not match, then, at 718, the Web server generates a new
23 one-way encrypted, confirmation session-state token while simulating that this
24 token generation is taking place at an earlier period of time. This can be simulated
25 by using data that identifies an earlier time bucket.

1 At 720, the Web server compares the new confirmation token with the
2 received token. If they match, then a new token is issued and sent to the client at
3 722. Issuing a new token can mean: specifying the most-recently-generated token
4 as the new token to be sent to the client; or generating a new token to be sent to
5 the client. After that, the user is allowed access the desired Web page or other
6 resources at 724.

7 If the new confirmation token and the received token do not match, then
8 one of two things may happen. Either the user will be forced to logon again at 726
9 or the process will loop back to block 718 where a newer confirmation token is
10 generated. A “newer” confirmation token is one that was more recently generated
11 than another confirmation token generated in the loop. Likewise, the “newest”
12 confirmation token is the one that was most recently generated of those generated
13 in the loop.

14 The process will perform this loop a specified number of times. Each
15 iteration of block 718 generates a newer confirmation token based upon a time
16 bucket earlier than the one used during the previous iteration. Each iteration of
17 block 720 compares the newest (i.e., most-recently-generated) confirmation token
18 with the received token.

19 If no match is made after looping a specified number of times, then the user
20 is forced to logon again at 726. This looping allows the current session of a user to
21 stay valid for up to X number of complete time buckets, where X is the specified
22 number of times that the steps are executed in the loop.

23 In the exemplary embodiment of the session-state manager, a Web page
24 checks to see if the user has logged in. The ASP of the Web page calls
25

1 CheckToken(). It passes in the input (e.g., UserID) and the token received from
2 the user.

3 If the user's session has expired or if the token is invalid, the component
4 returns a "timeout" indicator, which forces the user to logon again. Otherwise, it
5 returns the token, indicating that the session is valid. Periodically, the
6 CheckToken() function will return a refreshed token so that the session will expire
7 only if the user is inactive for a specified number of time buckets.

8 An exemplary section pseudocode for the CheckToken function is below
9 (the pseudocode is similar to C++ code for a COM implementation):

```
10     STDMETHODCALLTYPE CSessMan::CheckToken(BSTR root, BSTR token, BSTR  
11     *pbstrIsOK)  
12     {  
13  
14         HRESULT hr;  
15  
16         char logRoot[255];  
17         char logToken[255];  
18         char logTmpRoot[255];  
19         char logTmpRootReturn[255];  
20  
21         wcstombs(logToken, token, 255);  
22  
23         CComBSTR tmpRoot(root);  
24  
25         tmpRoot.Append("");  
26         if (tmpRoot.Length() == 0) {  
27             CComVariant vtOut("Timeout");  
28             hr = VariantChangeType(&vtOut, &vtOut, 0, VT_BSTR);  
29             if (FAILED(hr)) return hr;  
30  
31             *pbstrIsOK = ::SysAllocString(V_BSTR(&vtOut));  
32             return S_OK;  
33         }  
34  
35         wcstombs(logRoot, tmpRoot.m_str, 255);  
36  
37         char tmpOut[33];
```

```

1      tmpRoot.Append(m_szSecretSeed);
      tmpRoot.Append(GetTimeBucket());
2      wcstombs(logTmpRoot, tmpRoot.m_str, 255);
      CalcHash((char *)tmpRoot.m_str, tmpOut, ((int)(tmpRoot.Length()) + 1)
3      * 2);
      tmpOut[m_HashLength] = 0;
4      strcpy(logTmpRootReturn, tmpOut);
      if (CComBSTR(token) == CComBSTR(tmpOut)){
5          CComVariant vtOut(tmpOut);
          hr = VariantChangeType(&vtOut, &vtOut, 0, VT_BSTR);
6          if (FAILED(hr)) return hr;
          *pbstrIsOK = ::SysAllocString(V_BSTR(&vtOut));
7
          return S_OK;
8      }
      else{
9
10         char tmpOut2[33];
11
12         CComBSTR tmpRoot2(root);
13
14         tmpRoot2.Append(m_szSecretSeed);
15         tmpRoot2.Append(GetPrevTimeBucket());
16         CalcHash((char *)tmpRoot2.m_str, tmpOut2,
17         ((int)(tmpRoot2.Length()) + 1) * 2);
18         tmpOut2[m_HashLength] = 0;
19
20         if (CComBSTR(token) == CComBSTR(tmpOut2)){
21             CComVariant vtOut(tmpOut);
22             hr = VariantChangeType(&vtOut, &vtOut, 0, VT_BSTR);
23             if (FAILED(hr)) return hr;
24             *pbstrIsOK = ::SysAllocString(V_BSTR(&vtOut));
25
26             return S_OK;
27         }
28         else{
29             CComVariant vtOut("Timeout");
30             hr = VariantChangeType(&vtOut, &vtOut, 0, VT_BSTR);
31             if (FAILED(hr)) return hr;
32
33             *pbstrIsOK = ::SysAllocString(V_BSTR(&vtOut));
34
35             return S_OK;
36         }
37     }
38 }

```


TimeID Verification

Fig. 8 shows the steps of an exemplary alternative embodiment of the session-state manager that may be broadly called “TimeID verification.” Like the above-described embodiments of the session-state manager, this alternative exemplary embodiment is implemented by a Web server.

This exemplary embodiment may use the tokens described above, but not necessarily in the same way. Although it may use a token that is generated by an encryption scheme, it may also employ a token that is a plain numerical value corresponding to a specific time bucket. In addition, this exemplary implementation does not necessarily track user identification and logon validation information.

A Web server (which can be a single Web server as shown at 130 in Fig. 1 or a Web farm as shown at 240 of Fig. 2) generates a TimeID when the user initiates a session. This TimeID identifies the time bucket of the current time. The server sends that TimeID to the user via the client. Subsequently, the user sends that TimeID to the server.

Fig. 8 shows, at 810, a Web server receiving the TimeID that the user sent. That TimeID is now called the “user-associated TimeID”. At 812, the Web server designates a first TimeID that identifies a first time bucket. At 814, the Web server compares the first TimeID with the user-associated TimeID that it received in block 810.

If the two TimeIDs match, then access to the requested Web page of the server is allowed at 816. The two TimeIDs will match if both identify the same time bucket.

1 If the TimeIDs do not match, then, at 818, the Web server designates a prior
2 TimeID identifying a time bucket before the first time bucket. At 820, the Web
3 server compares the prior TimeID with the received TimeID. If they match, then
4 the first TimeID (for the first time bucket) is sent to the client at 822. After that,
5 the user is allowed access at 824.

6 If the TimeIDs do not match, then, at 818, the Web server designates a new
7 prior TimeID while simulating that this TimeID designation takes place at an
8 earlier time bucket. This can be simulated by using data that identifies an earlier
9 time bucket.

10 At 820, the Web server compares the new prior TimeID with the user-
11 associated TimeID. If they match, then a new TimeID is issued and sent to the
12 client at 822.

13 Issuing a new, user-associated TimeID can mean: specifying the most-
14 recently-designated TimeID as the new user-associated TimeID to be sent to the
15 client; or designating a new user-associated TimeID to be sent to the client. After
16 that, the user is allowed access the desired Web page or other resources at 824.

17 If the new prior TimeID and the user-associated TimeID do not match, then
18 one of two things may happen. Either the user will be forced to logon again at 826
19 or the process will loop back to block 818 where a newer prior TimeID is
20 generated.

21 The process will perform this loop a specified number of times. Each
22 iteration of block 818 generates a newer prior TimeID based upon a time bucket
23 earlier than the one used during the previous iteration. Each iteration of block 820
24 compares the newest (i.e., most-recently-designated) prior TimeID with the
25 received TimeID.

1 If no match is made after looping a specified number of times, then the user
2 is forced to logon again at 826. This looping allows the current session of a user to
3 stay valid for up to X number of complete time buckets, where X is the specified
4 number of times that the steps are executed in the loop.

5 In this exemplary alternative embodiment of the session-state manager, a
6 Web page checks to see if the user has logged in. These pages will typically be the
7 pages having limited-access information or other recourses. The ASP calls
8 function to check the TimeID. It passes in the seed (e.g., UserID) and the TimeID
9 received from the user. If the user's session has expired or if the TimeID is
10 invalid, the component returns a "timeout" indicator, which forces the user to
11 logon again. Otherwise, it returns the TimeID, indicating that the session is valid.
12 Periodically, the TimeID-checking function will return a refreshed TimeID so that
13 the session will expire only if the user is inactive for a specified number of time
14 buckets.

15 Again, this describes an alternative embodiment of the session-state
16 manager. This alternative embodiment employs non-encrypted tokens that tracks
17 only logon expiration. This alternative embodiment does not necessarily have a
18 high degree of security and it does not track user identification and logon
19 validation.

20 21 Conclusion

22 The main exemplary implementation described above and shown in Figs. 1-
23 7 uses a one-way encryption scheme. This provides a high degree of security and
24 reliability.
25

1 In a one-way encryption scheme, the door is locked and sealed shut. If an
2 interloper successfully unlocks and unseals the door, she would find only a key to
3 a safe-deposit box. To acquire the contents of the box, the interloper must select
4 the right bank, break into that bank, and find the box that the key fits.

5 Using this analogy, the encoded token is the key to the safe-deposit box; the
6 encrypted token is the key to and seal of the door behind which is placed the key;
7 and Web server is the bank housing the safe-deposit boxes including the one box
8 that the encoded token "safe-deposit" box unlocks. Inside, that box is session-
9 state information of the user.

10 However, there is one more level of security provided by the exemplary
11 implementation of the session-state manager. The Logon expiration of the
12 exemplary session-state manager is analogous to the bank changing the lock on the
13 safe deposit box if they haven't heard from their customer for a while.

14 Although the session-state manager has been described in language specific
15 to structural features and/or methodological steps, it is to be understood that the
16 session-state manager defined in the appended claims is not necessarily limited to
17 the specific features or steps described. Rather, the specific features and steps are
18 disclosed as preferred forms of implementing the claimed session-state manager.

CLAIMS

1. A session-state management method comprising:
generating an encoded session-state token, wherein the token incorporates a representation of session state of a client;
encrypting the encoded token using a one-way encryption scheme to produce an encrypted token; and
sending the encrypted token to the client.
2. A method as recited in claim 1, further comprising authenticating the user of the client.
3. A method as recited in claim 1, further comprising authenticating the user of the client, wherein the authenticating step comprises:
receiving a user identification indicator (“username”) and a password;
comparing the username to a database of authorized user records, each record containing a username and a username-associated password;
comparing the password received in the receiving step to a username-associated password of a record containing a matching username; and
establishing a session for the user.
4. A method as recited in claim 1, wherein the generating step comprises forming a confirmation token that incorporates a representation of an incremental time block.

1 5. A method as recited in claim 1, wherein the generating step
2 comprises forming a confirmation token that incorporates a representation of a
3 current incremental time block.

4
5 6. A method as recited in claim 1, wherein the generating step
6 comprises forming a confirmation token that incorporates a representation of an
7 incremental time block that is prior a current incremental time block.

8
9 7. A computer-readable storage medium having computer-executable
10 instructions that, when executed by a computer, performs the method as recited in
11 claim 1.

12
13 8. A session-state management method comprising:
14 receiving a one-way encrypted, session-state token from a client, wherein
15 the token incorporates a representation of session state of a client;
16 generating a one-way encrypted, confirmation session-state token;
17 comparing the confirmation token with the received token.

18
19 9. A method as recited in claim 8, wherein the generating step
20 comprises forming a confirmation token that incorporates a representation of an
21 incremental time block.

1 **10.** A method as recited in claim 8, wherein the generating step
2 comprises forming a confirmation token that incorporates a representation of a
3 current incremental time block.

4
5 **11.** A method as recited in claim 8, wherein the generating step
6 comprises forming a confirmation token that incorporates a representation of an
7 incremental time block that is prior a current incremental time block.

8
9 **12.** A method as recited in claim 8, further comprising:
10 issuing a one-way encrypted, replacement session-state token;
11 sending the replacement token to the client.

12
13 **13.** A method as recited in claim 12, wherein the issuing step comprises
14 forming a replacement token that incorporates a representation of a current
15 incremental time block.

16
17 **14.** A method as recited in claim 8, wherein the generating step
18 comprises forming a confirmation token that incorporates a representation of an
19 incremental time block, if confirmation and received tokens fail to match, the
20 method further comprising:

21 generating a new one-way encrypted, confirmation session-state token,
22 wherein the confirmation token incorporates a representation of a previous
23 incremental time block;

24 comparing the new confirmation token with the received token.
25

1 15. A method as recited in claim 14, wherein the new-confirmation-
2 token generating step comprises forming a confirmation token that incorporates a
3 representation of an incremental time block, if confirmation and received tokens
4 fail to match, the method further comprising:

5 repeating the steps of new-confirmation-token generating and comparing
6 the new and received tokens, wherein each subsequent reiteration of such steps
7 employs a representation of a previous incremental time block that is previous a
8 previous reiteration of the same steps, for a specified number of times or until
9 compared tokens match.

10
11 16. A computer-readable storage medium having computer-executable
12 instructions that, when executed by a computer, performs the method as recited in
13 claim 8.

14
15 17. A session-state management method comprising:

16 (A) receiving a one-way encrypted, session-state token from a client;

17 (B) generating a one-way encrypted, confirmation session-state token,
18 wherein the confirmation token incorporates a representation of a current
19 incremental time block;

20 (C) comparing the confirmation token with the received token;

21 (D) if the confirmation token and the received token match,

22 (1) issuing a one-way encrypted, replacement session-state token,
23 wherein the replacement token incorporates a representation of a current
24 incremental time block;

25 (2) sending the replacement token to the client.

1 if the confirmation token and the received token fail to match,

2 (3) generating a new one-way encrypted, confirmation session-state
3 token using the one-way encryption scheme of the encryption step,
4 wherein the token incorporates a representation of a previous incremental
5 time block;

6 (4) comparing the new confirmation token with the received token;

7 (5) if the new confirmation and received tokens fail to match, then
8 further comprising:

9 (i) repeating the steps of new-confirmation-token generating
10 and comparing the new and received tokens, wherein each
11 subsequent reiteration of such steps employs a representation of a
12 previous incremental time block that is previous a previous
13 reiteration of the same steps, for a specified number of times;

14 (ii) if, during the repeating step, the confirmation token
15 matches the received token,

16 (a) issuing a one-way encrypted, replacement session-
17 state token, wherein the token incorporates a representation
18 of a current incremental time block;

19 (b) sending the replacement token to the client.

20
21 **18.** A computer-readable storage medium having computer-executable
22 instructions that, when executed by a computer, performs the method as recited in
23 claim 17.
24
25

1 **19.** A session-state management method comprising:
2 authenticating a user of a client to establish a session with the user;
3 generating an encoded session-state token, wherein the encoded token
4 incorporates a representation of session-state of the user's session;
5 sending the session-state token to the client.

6
7 **20.** A method as recited in claim 19, wherein the authenticating step
8 comprises:

9 receiving a user identification indicator ("username") and a password;
10 comparing the username to a database of authorized user records, each
11 record containing a username and a username-associated password;
12 comparing the password received in the receiving step to a username-
13 associated password of a record containing a matching username; and
14 establishing a session for the user.

15
16 **21.** A method as recited in claim 19, wherein:
17 the user is identified by a user identification indicator (UserID);
18 the generating step comprises forming a session-state token at least partially
19 based upon the UserID.

20
21 **22.** A method as recited in claim 19, wherein:
22 a time block is identified by a time block identification indicator (TimeID);
23 the generating step comprises forming a session-state token at least partially
24 based upon the TimeID.
25

1 **23.** A method as recited in claim 19, wherein:
2 the user is identified by a user identification indicator (UserID);
3 a time block is identified by a time block identification indicator (TimeID);
4 the generating step comprises forming a session-state token at least partially
5 based upon the UserID and the TimeID.

6
7 **24.** A method as recited in claim 19, further comprising encrypting the
8 encoded token between the generating and the sending steps.

9
10 **25.** A method as recited in claim 19, further comprising one-way
11 encrypting the encoded token between the generating and the sending steps

12
13 **26.** A method as recited in claim 19, wherein:
14 the user is identified by a user identification indicator (UserID);
15 a time block is identified by a time block identification indicator (TimeID);
16 the generating step comprises combining UserID and TimeID to produce an
17 encoded token.

18
19 **27.** A computer-readable storage medium having computer-executable
20 instructions that, when executed by a computer, performs the method as recited in
21 claim 19.

1 **28.** A session-state token generation method, wherein an authenticated
2 user is identified by a user identification indicator (UserID) and a time block
3 identification indicator (TimeID) identifies a specific time block, the method
4 comprising:

5 combining UserID and TimeID to produce an encoded token;
6 encrypting the encoded token.

7
8 **29.** A method as recited in claim 28, wherein the combining step
9 comprises concatenating UserID and TimeID.

10
11 **30.** A method as recited in claim 28, wherein the combining step
12 comprises concatenating UserID, TimeID, and a code key.

13
14 **31.** A method as recited in claim 28, wherein the encrypting steps
15 comprises encrypting the encoded token using a one-way encryption scheme.

16
17 **32.** A method as recited in claim 28, wherein the encrypting steps
18 comprises:

19 encrypting the encoded token using a one-way encryption scheme to
20 produce an encrypted result; and

21 selecting a defined portion of the encrypted result to form a session-state
22 token.

1 **33.** A computer-readable storage medium having computer-executable
2 instructions that, when executed by a computer, performs the method as recited in
3 claim 28.

4
5 **34.** A session-state management method comprising:
6 receiving a user-associated, encoded session-state token from a client,
7 wherein the encoded token incorporates a representation of session-state of the
8 user's session;

9 generating an encoded, confirmation session-state token;
10 comparing the received token with the confirmation token.

11
12 **35.** A method as recited in claim 34, wherein the generating step
13 comprises forming a confirmation token that incorporates a representation of a
14 current incremental time block, if confirmation and received tokens fail to match,
15 further comprising:

16 generating a new confirmation token using a representation of a
17 incremental time block previous of the time block representation used for the
18 previous generating step;

19 comparing the new confirmation token with the received token.
20
21
22
23
24
25

1 **36.** A method as recited in claim 35, if confirmation and received tokens
2 fail to match, further comprising:

3 repeating the steps of generating a new confirmation token and comparing
4 the new and received tokens, wherein each subsequent reiteration of these steps
5 uses a representation of a previous incremental time block that is previous a
6 previous reiteration of the same steps, for a specified number of times or until
7 compared tokens match.

8
9 **37.** A method as recited in claim 34, wherein the user-associated
10 session-state token is encrypted.

11
12 **38.** A method as recited in claim 34, wherein the user-associated
13 session-state token is one-way encrypted.

14
15 **39.** A computer-readable storage medium having computer-executable
16 instructions that, when executed by a computer, performs the method as recited in
17 claim 34.

18
19 **40.** A session-state management method comprising:
20 receiving a user-associated TimeID from a client, wherein the encoded
21 token incorporates a representation of session-state of the user's session;
22 designating a first time block identification indicator (TimeID) for a first
23 time block;
24 comparing the user-associated TimeID with the first TimeID.
25

1
2 **41.** The method of claim 40, further comprising:

3 designating a prior TimeID for a time block prior to the first time block;

4 comparing the user-associated TimeID with the prior TimeID.

5
6 **42.** A server to communicate with a client over a communications
7 network, the server comprising:

8 a processor;

9 a session-state manager executable on the processor to:

10 generate a session-state token, wherein the token incorporates a
11 representation of session state of the client;

12 encrypt the token using a one-way encryption scheme to produce an
13 encrypted token; and

14 send the encrypted token to the client.

15
16 **43.** A server to communicate with a client over a communications
17 network, the server comprising:

18 a processor;

19 a session-state manager executable on the processor to:

20 receive a one-way encrypted, session-state token from the client,
21 wherein the token incorporates a representation of session state of a client;

22 generate a one-way encrypted, confirmation session-state token;

23 compare the confirmation token and the received token.
24
25

1 **44.** A server to communicate with a client over a communications
2 network, the server comprising:

3 a processor;

4 a session-state manager executable on the processor to:

5 authenticate a user of the client;

6 generate an encoded session-state token, wherein the token
7 incorporates a representation of session state of the client; and

8 send the session-state token to the client.

9
10 **45.** A manager as recited in claim 44, wherein session-state manager
11 executable on the processor to encrypt the encoded session-state token.

12
13 **46.** A manager as recited in claim 44, wherein session-state manager
14 executable on the processor to one-way encrypt the encoded session-state token.

15
16 **47.** A server to communicate with a client over a communications
17 network, wherein an authenticated user is identified by a user identification
18 indicator (UserID) and a time block identification indicator (TimeID) identifies a
19 specific time block, the server comprising:

20 a processor;

21 a session-state manager executable on the processor to:

22 combine UserID and TimeID to produce an encoded token; and

23 encrypt the encoded token.

1 **48.** A server to communicate with a client over a communications
2 network, the server comprising:

3 a processor;

4 a session-state manager executable on the processor to:

5 receive a user-associated, encoded session-state token from the
6 client;

7 generate an encoded, confirmation session-state token, wherein the
8 confirmation token incorporates a representation of session state of the
9 client;

10 compare the received token with the confirmation token.

11
12 **49.** A computer-readable storage medium having computer-executable
13 instructions that, when executed by a computer, performs the method comprising:

14 generating an encoded session-state token, wherein the token incorporates a
15 representation of session state of a client;

16 encrypting the encoded token using a one-way encryption scheme;

17 sending the encrypted token to the client.

18
19 **50.** A computer-readable storage medium having computer-executable
20 instructions that, when executed by a computer, performs the method comprising:

21 receiving a one-way encrypted, session-state token from a client, wherein
22 the token incorporates a representation of session state of a client;

23 generating a one-way encrypted, confirmation session-state token;

24 comparing the confirmation token with the received token.
25

ABSTRACT

Without actually storing session-state information, the described exemplary implementations of session-state manager identify a user, validate the user's current logon state, and determine whether the user's session should expire. User identification and logon validation are checked by a server in a stateless network by generating a mathematically session-state token and sending that token to a user. Subsequently, the server receives a mathematically session-state token from the user and checks that token. If that token checks out, then the user is allowed continuing access under the same session. If it doesn't check out, then the user may be forced to start a new session by logging-on again. Alternatively, the server may check to see if the token would check out if it had come at an earlier time block. The session-state tokens are mathematical encoded and are generated using a one-way encryption scheme. Such a one-way encrypted token is scientifically impossible to reverse-engineer. Furthermore, logon expiration is checked by the server using the same mathematically session-state token. The token is checked to determine whether a predetermined number of time blocks have past. If so, then the server will terminate the user's session.

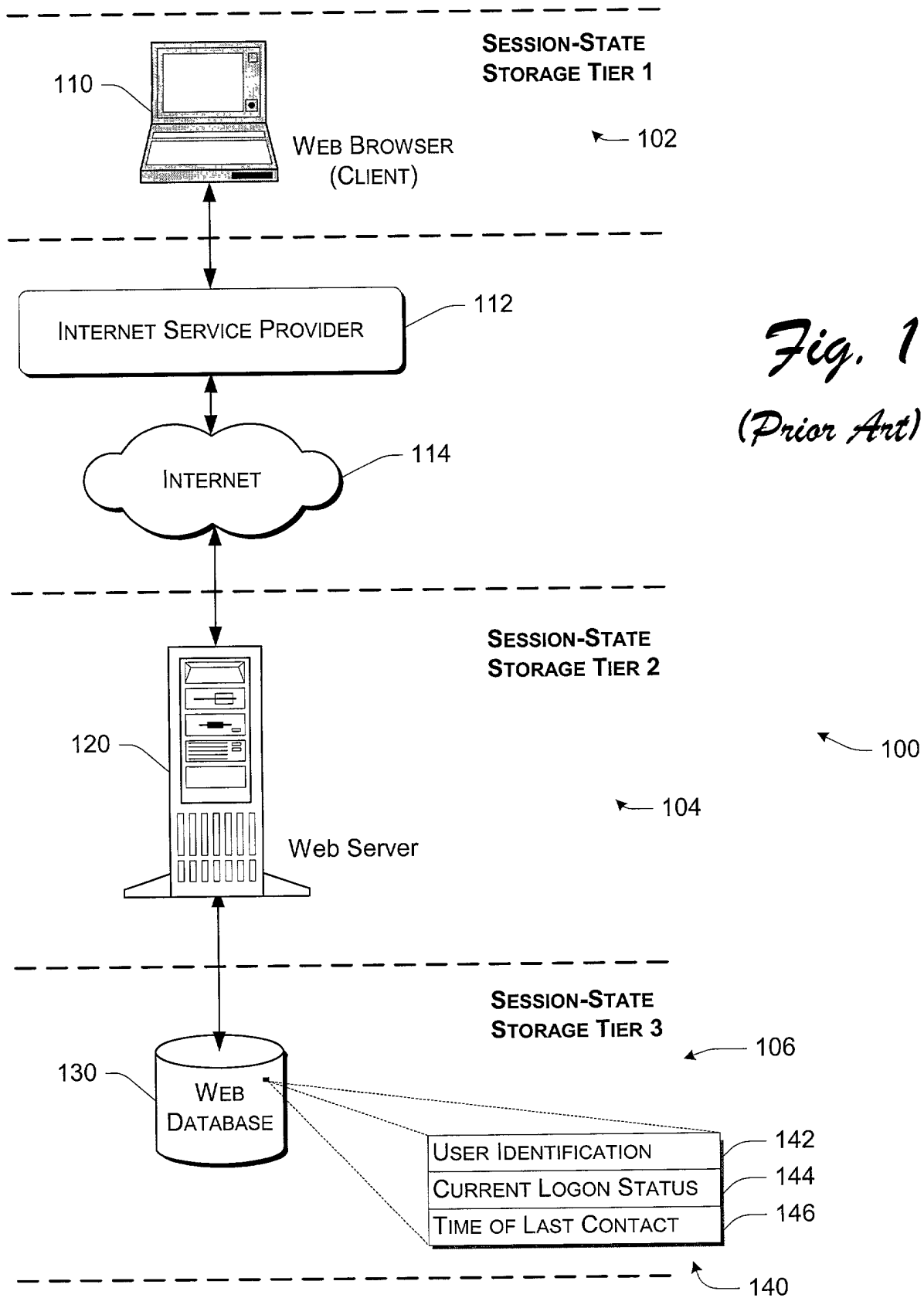


Fig. 2

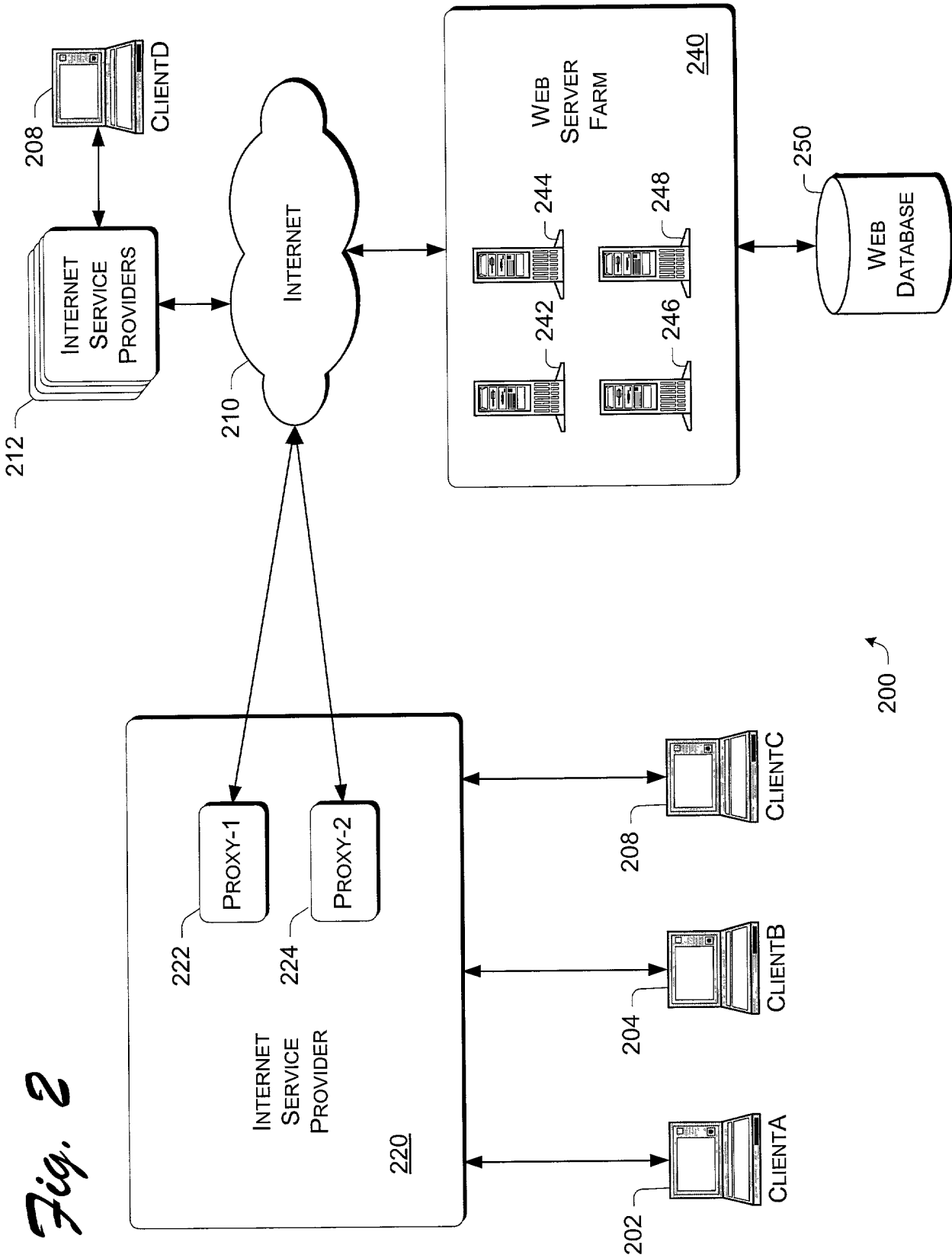
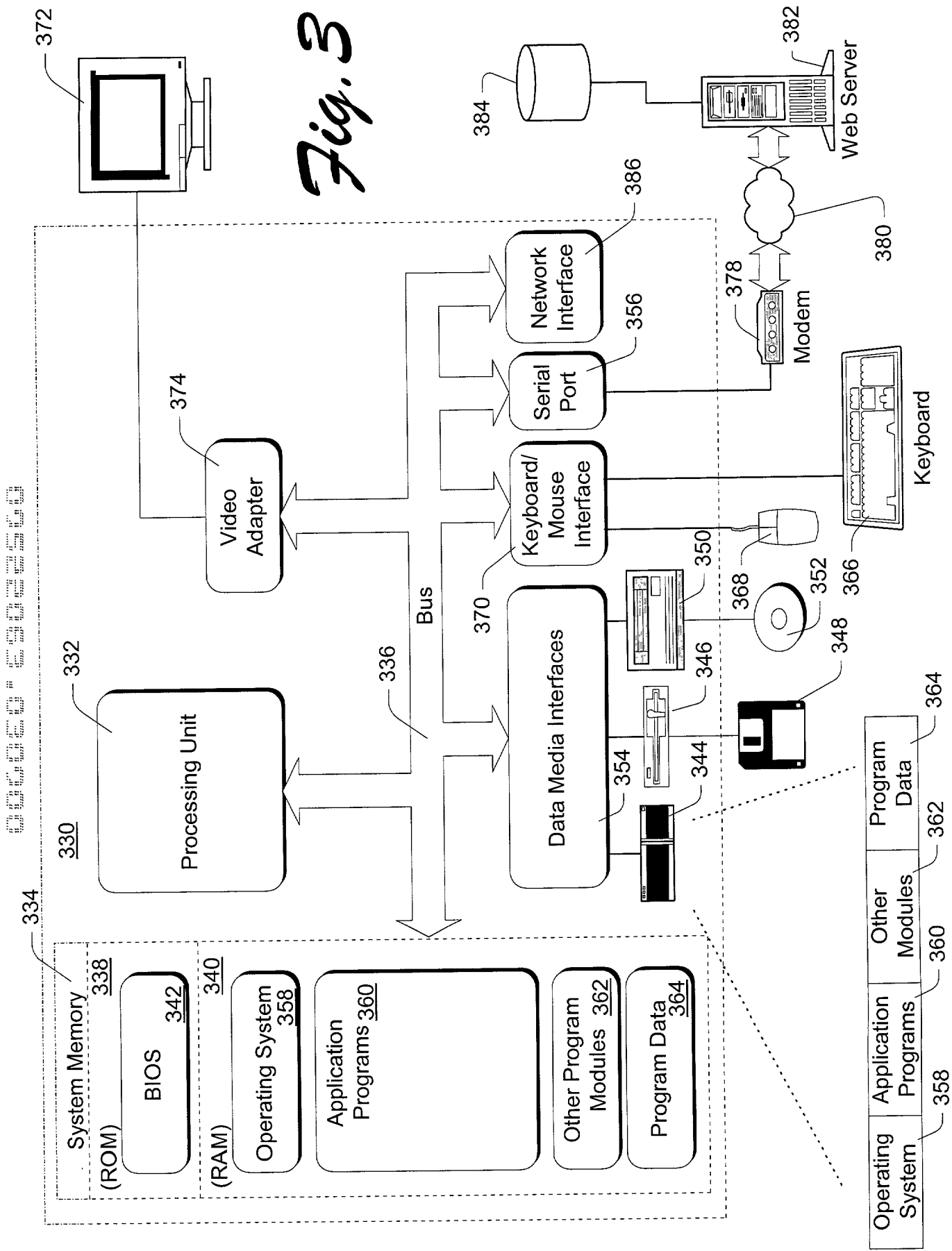


Fig. 3



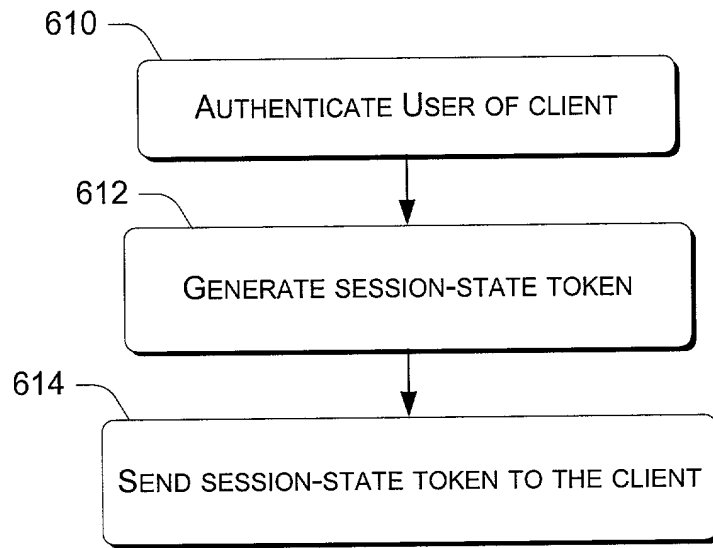


Fig. 6

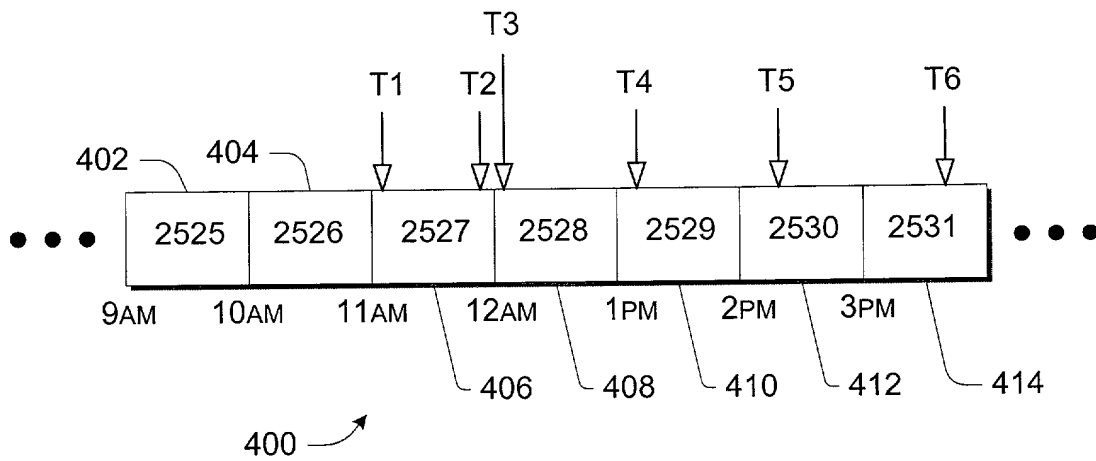


Fig. 4

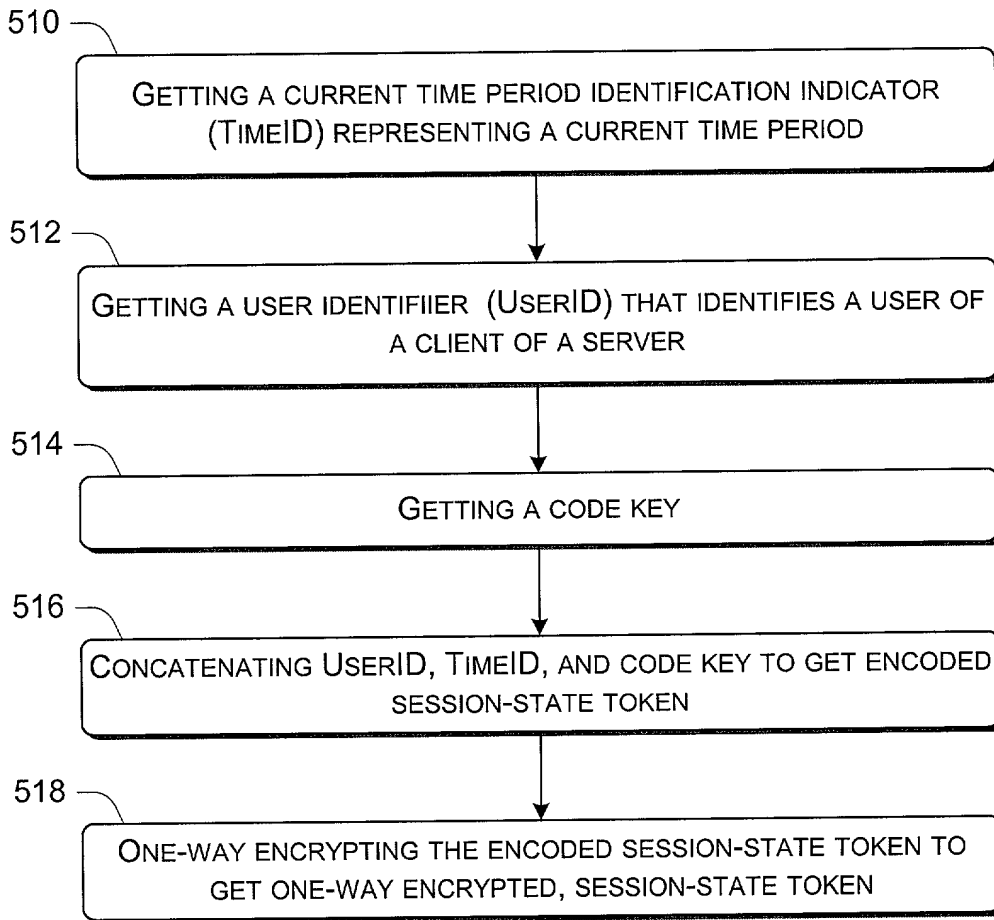
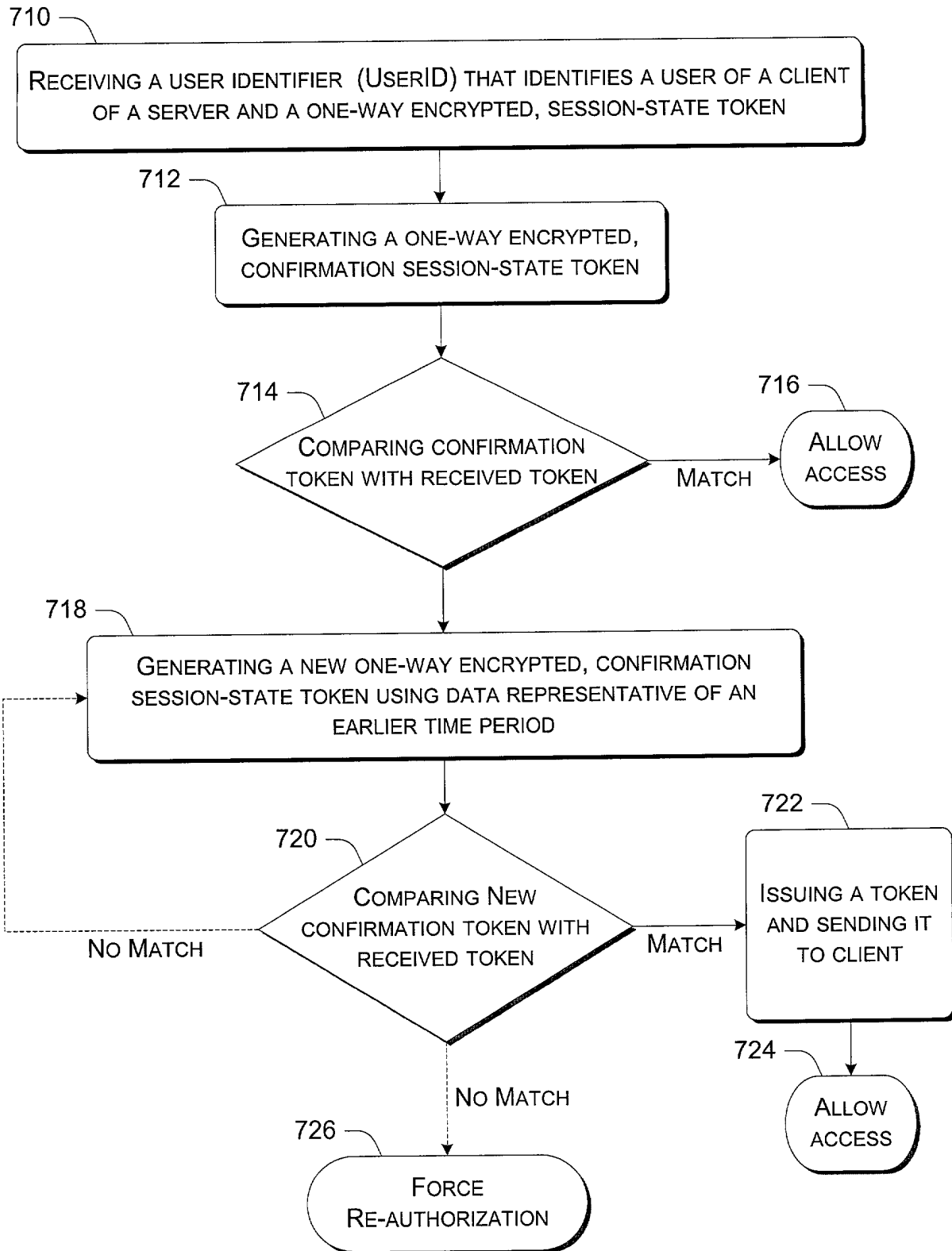
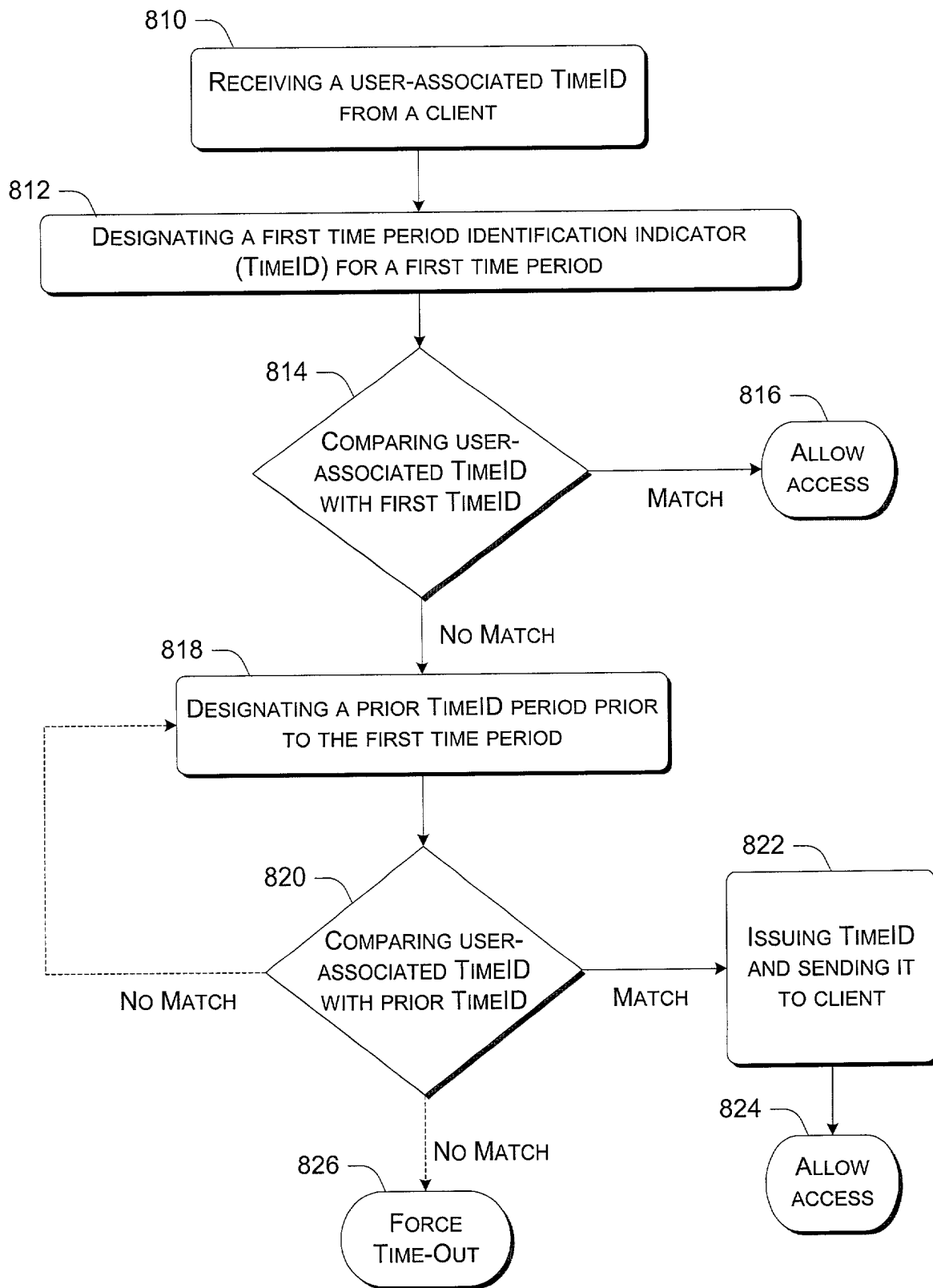


Fig. 5

*Fig. 7*

*Fig. 8*